# Introduction

## Généralité

|  |  |  |
| --- | --- | --- |
| **Projet** | | |
| **Nom du projet** | EasyGoing | |
| **Début du projet** | 14.09.2015 | |
| **Fin du projet** | 04.01.2016 | |
| **Durée effective** | 90 heures | |
| **Acteurs** | | |
| **Répondant** | Dr. René Rentsch | rene.rentsch@heig-vd.ch |
| **Développeurs** | Miguel Santamaria (Chef de projet)  Raphaël Racine (Suppléant)  Thibaud Duchoud  Michelle Vanessa Meguep Sakam  Karim Ghozlani | [miguel.santamaria@heig-vd.ch](mailto:miguel.santamaria@heig-vd.ch)  [raphael.racine@heig-vd.ch](mailto:raphael.racine@heig-vd.ch)  [thibaud.duchoud@heig-vd.ch](mailto:thibaud.duchoud@heig-vd.ch)  [michelle.meguepsakam@heig-vd.ch](mailto:michelle.meguepsakam@heig-vd.ch)  karim.ghozlani@heig-vd.ch |

# Objectifs du projet

Ce projet a pour objectif le développement d’un site web permettant la gestion de projets.

Il permet aux membres d’une équipe de gérer un projet autour d’un « scrum board ». Il offre la possibilité de visualiser différentes tâches, qui travaille sur l’une de ces dites tâches et l’avancement de celle-ci.

Des événements ont été mis en place et ceux-ci permettent de suivre les différentes actions effectuées par un utilisateur (par exemple lors du déplacement d’une tâche ou de l’ajout d’un membre au projet). Ce système est conçu comme un fil d’actualité.

Notre projet dispose également d’un tutoriel d’utilisation qui facilite grandement la tâche des nouveaux venus et d’une grande ergonomie

# Technologies utilisées et conventions de codage

### Technologies

Voici la liste des technologies utilisées :

* **Apache v2.4** comme serveur http.
* **PHP v5.6** comme langage coté serveur.
  + **Zend Framework 2** comme framework php.
* **MySQL v5.6** comme moteur de base de données.
  + **PHPMyAdmin v4.4 et MySQL Workbench v6.3** pour gérer la base de données.
* **HTML – CSS** comme langages de structure et design des pages web.
* **Javascript/JQuery v2.1** comme langage/bibliothèque pour dynamiser les pages web.
* **Différentes bibliothèques Javascript** (comme Bootstrap) pour réaliser des actions spécifiques concernant la gestion des boards principalement (cliquer-déposer, animation, gestion d’une grille …).

### Conventions de codage

#### Général

Les conventions de codage que nous avons utilisées sont inspirées de celles utilisées dans le framework Zend et qui sont disponibles sur internet (<http://framework.zend.com/manual/1.12/fr/coding-standard.naming-conventions.html>).

Tout doit être écrit en anglais.

#### Nommage

Les noms de fonctions, de variables et de classes doivent être descriptifs : il faut éviter les abréviations.

Les noms des fichiers PHP correspondent à la classe qu’il contient.

Les variables et fonctions sont écrites en camelCase (première lettre en minuscule et chaque nouveau mot commence par une majuscule).

Pour les variables de classes (membres), si elles sont « protected » ou « private » ont les précède d’un « \_ ».

Les méthodes respectent les mêmes conventions que les variables de classes.

Les constantes sont écrites en majuscule et chaque mot est séparé par un « \_ ».

Les classes sont écrites en PascalCase (chaque mot commence par une majuscule).

#### Autres

En PHP, la première accolade suivant une instruction se place après un retour à la ligne. En Javascript, la première accolade se met directement après l’instruction.

L’indentation se compose de 3 espaces (donc pas de tabulation).

# Rôle du contrôleur « ProjectController »

Le contrôleur « ProjectController » est relatif à un projet en particulier. Il gère toutes les actions qui sont relatives à un projet. Il va, entre autres, afficher les informations relatives à un projet, les détails d’une tâche qui se trouve dans un projet, la liste des membres qui peuvent rejoindre ce projet. Il gère également l’ajout, suppression d’un membre dans le projet et l’ajout, suppression et édition d’une tâche ou sous-tâche dans le projet.

Voici les différentes actions qui le composent :

1. indexAction\* : cette action s’occupe de récupérer les différentes informations relatives au projet courant (le projet, les tâches et sous-tâches qui s’y trouvent, les membres, les événements et d’autres informations utilisées dans la vue). Elle va ensuite demander à la vue d’afficher ces différentes informations.
2. editAction\* : cette action va mettre à jour les informations d’un projet, après une édition de celui-ci. Elle réagit différemment selon si l’utilisateur a effectué une requête POST (c’est-à-dire qu’il a appuyé sur le bouton « Save changes ») ou non :
   1. Dans le premier cas, elle validera les différents champs entrés par l’utilisateur et les droits de celui-ci. Si les champs sont corrects et que l’utilisateur a les droits nécessaires pour modifier les propriétés d’un projet, elle met à jour les valeurs dans la base de données et redirige l’utilisateur sur la page d’index des projets ; s’ils ne sont pas valides ou qu’il n’a pas les droits, elle renvoie une erreur.
   2. Dans le second cas, l’utilisateur n’a pas fait de requête POST et donc elle appelle simplement la vue qui affichera le formulaire d’édition d’un projet.
3. addTaskAction\* : cette action s’occupe d’ajouter une nouvelle tâche au projet. Cette action va, comme la précédente, faire deux choses lorsqu’elle reçoit une requête POST ou non. Dans les 2 cas on va vérifier, si la tâche qu’on veut ajouter est une sous-tâche, que celle-ci est valide.
   1. Dans le cas d’une requête POST, elle vérifie les champs entrés par l’utilisateur. S’ils sont corrects on ajoute la tâche dans la base de données et, si ce n’est pas une sous-tâche, on l’affecte à son créateur. Si les champs sont incorrects elle renvoie une erreur.
   2. Si ce n’est pas une requête POST, elle va afficher le formulaire permettant l’ajout d’une tâche.
4. taskDetailsAction\* : cette action va afficher les détails d’une tâche. Elle récupère les différentes informations à afficher (les informations relatives à une tâche) et les envoie à la vue qui va nous être affichées.
5. editTaskAction\* : cette action s’occupe de l’édition d’une tâche. Elle fait également deux choses lorsqu’elle reçoit une requête POST ou non.
   1. Dans le cas d’une requête POST, elle vérifie les champs entrés par l’utilisateur. S’ils sont corrects elle met à jour les informations dans la base de données et redirige l’utilisateur sur la page d’index d’un projet.
   2. Dans le second cas, elle affiche la vue d’édition d’un projet en passant les informations relatives au projet qui y seront affichées.
6. boardViewMembersAction : cette action va retourner les informations relatives au « board » qui affiche les membres dans la colonne de gauche. Elle va récupérer les membres du projet (et des informations les concernant), leurs spécialisations et les tâches affectées à chaque membres. Elle va ensuite créer une vue en lui envoyant les informations récupérées et elle va enlever le « layout » de cette vue car celle-ci est chargée, du côté client, en ajax (requête « load »)
7. boardViewTasksAction : cette action va retourner les informations relatives au « board » qui affiche les tâches dans la colonne de gauche. Elle va récupérer les tâches principales (celles qui ne sont pas des sous-tâches) du projet et les membres affectés à chaque tâche. Elle va ensuite faire la même chose que l’action « boardViewMembersAction » mais c’est une autre vue qui est créée.
8. assignTaskAction\* : cette action gère l’assignement d’une tâche à un membre d’un projet. Elle est appelée lorsqu’un utilisateur « drag-drop » une tâche non assignée pour la mettre dans le « board » (il assigne la tâche à un membre). Les tâches sont assignées en déplaçant les liens qui se trouvent en dessous du « board » dans la liste des tâches du projet.

Elle va vérifier les droits de l’utilisateur et si la tâche est déjà assignée à l’utilisateur cible ou non. Dans le cas où l’utilisateur a les droits nécessaires et la tâche n’est pas déjà assignée à l’utilisateur cible, on ajoute une affectation dans la base de données. Sinon elle affiche une erreur.

1. moveTaskAction\* : cette action gère le déplacement d’une tâche dans le « board ». Lorsqu’un utilisateur déplace une tâche du « board » vers un autre utilisateur ou dans une autre colonne.

Elle va également vérifier les droits de l’utilisateur et, si l’utilisateur déplace la tâche vers un autre utilisateur, que celle-ci ne lui est pas déjà assignée. Dans le cas où tout est ok, elle met à jour les informations dans la base de données. Sinon elle affiche une erreur.

1. unassignTaskAction\* : cette action gère la désaffectation d’une tâche à un utilisateur. Elle est appelée lorsqu’un utilisateur fait un clic droit sur une tâche et sélectionne « Unassign ».

Elle va vérifier les droits de l’utilisateur et s’il a les droits, elle supprime l’affectation correspondante dans la base de données. Sinon elle affiche une erreur.

1. deleteTaskAction\* : cette action gère la suppression d’une tâche. Elle est appelée lorsqu’un utilisateur fait un clic droit sur une tâche et sélectionne « Delete ».

Elle va vérifier les droits de l’utilisateur et si ceux-ci sont suffisants elle supprime la tâche de la base de données et les affectations correspondantes. Sinon elle affiche une erreur.

1. addMemberAction\* : cette action gère l’ajout d’un utilisateur dans un projet. Elle a deux comportements différents en fonction de si la requête reçue est de type POST ou non. Quel que soit le type de requête, elle va vérifier les droits de l’utilisateur et si celui-ci n’a pas les bons elle le redirige vers la page d’index du projet.
   1. Dans le premier cas, elle va récupérer les informations postées et celles-ci correspondent au(x) utilisateur(s) à ajouter. Elle va parcourir ses informations et ajouter chaque utilisateur au projet.
   2. Dans le second cas, elle affiche la vue qui permet d’ajouter des utilisateurs au projet (affiche la liste des utilisateurs qui ne sont pas membres du projet et l’utilisateur peut les sélectionner afin de les ajouter).
2. removeMemberAction\* : cette action gère le fait de retirer un membre d’un projet. Elle va vérifier les droits de l’utilisateur et s’ils sont suffisants elle va mettre à jour la base de données en supprimant l’affectation du membre au projet et les spécialisations relatives à celui-ci.
3. detailsAction : cette action retourne, sous format JSON, les informations relatives à un projet. Elle est utilisée lorsqu’un déroule un projet dans la liste des projets.
4. postNewFeedAction\* : cette action gère l’ajout d’une nouvelle « news » correspondant à une tâche. Quand on est sur la page qui affiche le détail d’une tâche l’utilisateur à la possibilité d’ajouter une « news » sur cette tâche. Cette action est appelée lorsqu’un utilisateur ajoute une nouvelle « news » sur une tâche. Elle va récupérer les informations entrées par l’utilisateur et les ajouter dans la base de données. Elle retourne un JSON contenant un champ « success » qui est à « true » s’il n’y a pas d’erreur et à « false » sinon. Lorsque la requête reçue n’est pas de type POST elle retourne se même JSON avec le champ « success » à « false ».

\* Les actions suivies d’un astérisque (\*) contiennent toutes une gestion des événements. Pour chacune de ces tâches, des événements sont créés et insérés dans la base de données. Ces événements correspondent à ce qui est fait dans la tâche en question. Par exemple lorsqu’on retire un membre du projet (*removeMemberTask*) un événement de type utilisateur est créé avec la date courante et contenant le texte : « xxx removed user yyy from project, bye bye ! ». Le texte et le type d’événement change, bien évidemment, en fonction de l’action appelée.

Ces actions envoient, pour chaque événement, une requête au serveur qui gère les événements afin que les autres utilisateurs présent sur le projet en soit informé en temps réel.

(Pour plus d’informations c.f. le chapitre « Serveurs secondaires d’événements »).

# Fonctionnement du « Dash board »